(NeXT Tip #47) Shared Libraries

Christopher Lane (*lane[at]CAMIS.Stanford.EDU*)
Wed, 2 Feb 1994 13:39:06 -0800 (PST)

Traditional Unix 'static' code libraries (lib*.a) provide useful routines that get compiled into every program that uses them. A feature of OS on our NeXT, Sun & HP systems is the use of 'shared libraries' that contain object code that several executable files can use simultaneously.

When a program is compiled with a shared library, the code that implements the external routines (like strcmp(), random(), etc.) is not copied into the object file and data is copied only if referenced directly by the program. The program binds addresses of routines in a branch table which gets filled in to point to actual code. A dynamic loader fills in the branch table entries at run-time once the actual libraries have been located in the file system.

Shared libraries save disk space because programs don't contain copies of the archive of common routines. They also save memory because all running programs point to one shared copy of the code. (The first application that touches a shared library pulls it into memory -- all others after that point to that copy in memory.) A shared library also helps to keep applications up to date, because when replaced, all applications use the new version without needing to be recompiled.

The startup cost of dynamic loaders can be significant, even with deferred binding, and can cause performance degradation in programs dominated by startup (eg. simple 'hello world' program). Some position-independent shared library code can also be slower than normal code. However, the advantages of decreased disk space usage and decreased memory requirements for executables usually outweigh these problems.

Unfortunately, it's harder to maintain a shared library because more things need to remain constant in order for new versions to be compatible. Most shared library implementations include version control to detect potential incompatibilities. (You may have seen 'ld.so: warning: libX has older revision than expected 2.3' errors on a Sun.)

On our NeXT and Sun systems, there are two parts to a shared library, the compile-time (or 'host') library and the run-time (or 'target') library. The compile-time library has a name like lib*_s.a on a NeXT and lib*.sa.X.X on a Sun, where X.X is the version information. The run-time library must be installed on the machine in order for the application to execute as it contains the branch table and actual code. It has a name like lib*_s.X.shlib on a NeXT (where X is a version letter like A, B, C, etc.) or lib*.so.X.X on Sun. HP combines the host and target function into a single lib*.sl file.

On the NeXT, 'otool -L' shows which run-time libraries a program expects:

> otool -L ArchiveBrowser.app/ArchiveBrowser
ArchiveBrowser.app/ArchiveBrowser:
/usr/shlib/libMedia_s.A.shlib (minor version 12)
/usr/shlib/libNeXT_s.C.shlib (minor version 57)
/usr/shlib/libsys_s.B.shlib (minor version 55)

Although they all use a similar dynamic loading scheme, the three systems vary slightly in how they locate shared libraries at run-time. NeXT and HP include in the executable the full path name to the actual library -- although HP has various options at compile-time to allow this path to differ from what was compiled against. Sun uses a file, /etc/ld.so.cache, to map between a library name, 'phigs', and the actual library, '/usr/lib/libphigs.so.2.1' -- this cache is kept up to date by occasional runs of the ldconfig utility.

The Sun scheme allows you to keep older versions of libraries available as

separate files. I believe HP's library format and version control system allow you to keep multiple versions of the same routine in a library when incompatible changes are made to a library interface. The hard coded path in the NeXT binaries has been an issue when trying to get old, incompatible programs like Icon, Scene & ACL to run in newer releases -- at times requiring manually editing binaries to point to directories containing older libraries. (Actually a problem with proper update of version numbers by the vendor.)

There are various flags to 'ld' to create shared libraries and control their behavior. For example, on the HP there's an option to ld to tell the dynamic loader to bind all necessary references at startup. This dramatically increases the startup time of a program, but ensures that no more dynamic binding operations will be required later for better real time response.

The use of shared libraries doesn't necessarily eliminate traditional, static libraries. The NeXT & Sun systems have about a dozen shared system libraries (X windows adds another dozen) but dozens more standard, static lib*.a libraries. On these systems, shared libraries are generally large collections of routines that will be in use by multiple programs. The HP systems uses many more shared libraries -- this may reflect dynamic loader overhead and other tradeoffs. For some shared libraries, there are also static versions available for special situations.

On the Sun & HP systems, you can find both traditional and shared libraries on /usr/lib, among other places. On the NeXT, traditional libraries and compile-time libraries can be also be found on /usr/lib (& elsewhere) but the run-time components of shared libraries are found on /usr/shlib. (And /usr/sh2.1 on our NeXTs as that's where older versions of libraries for patched binaries are kept.)

For more about shared libraries, see the man pages for 'ld' on all the systems as well as the the man pages for 'ld.so' & 'ldconfig' on the Sun, 'dld.sl' on the HP and 'otool' on the NeXT as well as 1447_what_are_shared_libraries.rtf in NextAnswers (all of which were sources for most of the above information).

- Christopher